Module4: Collections Framework, Event Handling

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

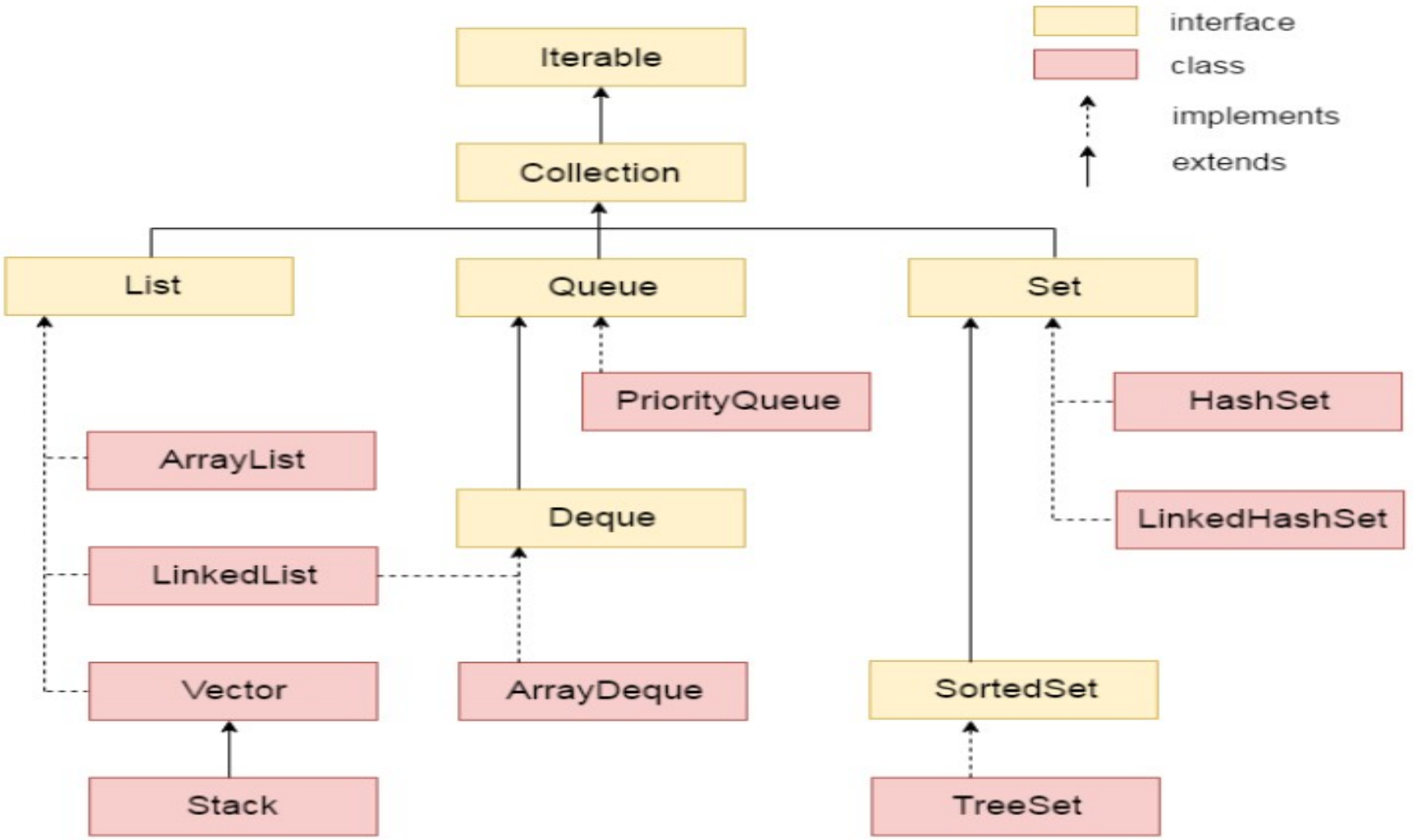
What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework. The **java.util** package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	public int size()	It returns the total number of elements in the collection.
6	public void clear()	It removes the total number of elements from the collection.
7	public boolean contains(Object element)	It is used to search an element.
8	public boolean containsAll(Collection c)	It is used to search the specified collection in the collection.
9	public Iterator iterator()	It returns an iterator.
10	public Object[] toArray()	It converts collection into array.
11	public boolean isEmpty()	It checks if collection is empty.
12	default Spliterator<E> spliterator()	It generates a Spliterator over the specified elements in the collection.
13	public boolean equals(Object element)	It matches two collections.

1. ArrayList class

ArrayList is a part of collection framework and is present in java.util package.
The **ArrayList** class extends **AbstractList** and implements the **List** interface.
ArrayList is a generic class that has this declaration:

class ArrayList<E>

Here, **E** specifies the type of objects that the list will hold.
ArrayList supports dynamic arrays that can grow or shrink as needed.

In Java, standard arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold.

But, sometimes, you may not know until run time precisely how large an array you need. To handle this situation, the Collections Framework defines **ArrayList**. An **ArrayList** is a variable-length array of object references. That is, an **ArrayList** can dynamically increase or decrease in size.

Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

- Can contain duplicate elements.
- Maintains insertion order.
- Non synchronized.
- Allows random access because array works at the index basis.
- Manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.

Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of Java ArrayList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list.
void clear()	It is used to remove all of the elements from this list.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.
void remove(Object o)	It is used to remove a matched object
void remove(int index)	It is used to remove the object at specified index
int size()	Returns the number of elements in the collection

void removeAll(Collection c)	Removes the specified collection from the list
Object get(int index)	Returns the element specified in the index
void set(int index,Object o)	Replace the element in index with specified object
boolean contains(Object)	checking if the specified element exists in the given list or not.
boolean isEmpty()	Returns true if this list contains no elements.
Spliterator<E> spliterator()	It is used to create spliterator over the elements in a list.
List<E> subList(int start, int end)	It is used to fetch all the elements lies within the given range.
protected void removeRange(int start, int end)	It is used to remove all the elements lies within the given range.

```
// Demonstrate ArrayList.
import java.util.*;
class ArrayListDemo
{
public static void main(String args[])
{
//ArrayList al = new ArrayList();
//non-generic
ArrayList<String> al = new
ArrayList<String>(); //generic type
System.out.println("Initial size of al: "
+al.size());
// Add elements to the array list.
al.add("CHENNAI");
al.add("AHMEDABAD");
al.add("BANGALORE");
al.add("HYDERABAD");
al.add("DELHI");
al.add("PATNA");
//displaying list
System.out.println( al);
System.out.println("After adding at a
position ");
al.add(3, "KOLKATA");
System.out.println("Size of ArrayList
after additions: " +al.size());
// Display the array list.
System.out.println( al);
```

```
System.out.println("The index of
BANGALORE "
+al.indexOf("BANGALORE"));
// Remove elements from the array list.
al.remove("DELHI");
al.remove(2);
al.add(5, "HYDERABAD");
//the array list using for-each
System.out.println("The Array List
is");
for(String name : al)
System.out.println(name);
System.out.println("Thelast
indexofHyderabad"
+al.lastIndexOf("HYDERABAD"));
ArrayList<String> bl = new
ArrayList<String>();
bl.add("MUMBAI");
bl.add("BHOPAL");
al.addAll(4,bl);
if(al.contains("BHOPAL"))
System.out.println("Bhopal Exist");
if(al.containsAll(bl))
System.out.println("Mumbai Bhopal
Exist");
System.out.println("Array list after
adding another list ");
System.out.println(al);
System.out.println("Array list after
deleting new list ");
al.removeAll(bl);
System.out.println(al);
al.clear();
System.out.println("Size of array list
after clear: " +al.size());
System.out.println("The elements
after clear: " +al);
```

```
// Demonstrate ArrayList with
Userdefined Objects.
import java.util.*;
class Book
{
int id;
String name,author;
int quantity;
public Book(int id, String name,
String author, int quantity)
{
this.id = id;
this.name = name;
this.author = author;
this.quantity = quantity;
} }
public static void main(String[] args)
{
//Creating list of Books
ArrayList<Book> list =new
ArrayList<Book>();
```

```
public class ArrayListDemo1
{
//Adding Books to list
list.add(b1);
list.add(b2);
list.add(b3);
//Creating Books
Book b1=new Book(101,"Let us
C","Yashwant Kanetkar",8);
Book b2=new Book(102,"Data
Communications","Forouzan",4);
Book b3=new Book(103,"Operating
System","Galvin",6);
```

```
// Demonstrate ArrayList with
Userdefined Objects.
import java.util.*;
class Book
{
int id;
String name,author;
int quantity;

public Book(int id, String name,
String author, int quantity)
{
this.id = id;
this.name = name;
this.author = author;
this.quantity = quantity;
} }
public class ArrayListDemo1
{ public static void main(String[] args) {
//Creating list of Books
ArrayList<Book> list =new
//Creating Books
Book b1=new Book(101,"Let us
C","Yashwant Kanetkar",8);
Book b2=new Book(102,"Data
Communications","Forouzan",4);
Book b3=new Book(103,"Operating
System","Galvin",6);
//Adding Books to list
list.add(b1);
list.add(b2);
list.add(b3);
```

```
// Demonstrate ArrayList with
Iterator.
import java.util.*;
public class ArrayList3
{
public static void main(String[] args)
{ List<String> ls = new
ArrayList<String>();
ls.add("Varsha");
ls.add("Harsha");
ls.add("Srihrasha");
ls.add("Srivarsha");
ls.add("harshith");
//Accessing ArrayList using Iterator
Iterator i = ls.iterator();
System.out.println("The ArrayList
elements are:");
while (i.hasNext())
{
```

```
// Demonstrate ArrayList with
Spliterator.
import java.util.*;
public class ArrDemo4
{
public static void main(String[] args)
{ ArrayList<String> list = new
ArrayList<String>();
list.add("Hyderabd");
list.add("Pune");
list.add("Bangalore");
list.add("Kolkata");
list.add("Chennai");
```

```
// Demonstrate ArrayList with
Userdefined Objects.
public class EmpArrayList
{ public static void main(String[] args)
{ //Creating list of Employees
ArrayList<Employee> list =new
ArrayList<Employee>();
//Creating Employees
Employee e1=new
Employee(101,"A.Harsha",75000.50);
Employee e2=new
Employee(102,"B.Varsha",85000.50);
Employee e3=new
Employee(103,"C.Sirisha",95000.50);
Employee e4=new
Employee(104,"D.Sandeep",195000.5
0);
//Adding Employees to list
list.add(e1); list.add(e2); list.add(e3);
list.add(e4);
//Displaying Number of Employees
//Displaying Details of Employees
System.out.println("\n The employess
data is \n");
for(Employee e:list)
{
System.out.println(e.eid+":"+e.ename+":
"+e.sal);
System.out.println();
}
//Deleting an Employee
list.remove(2);
System.out.println("\n After
removing number of employees are"
+ list.size());
//Displaying Details of Employees
System.out.println("\n The employess
```

2. LinkedList class

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

- **Can contain duplicate elements.**
- **Maintains insertion order.**
- **Non synchronized.**
- **Manipulation is fast because no shifting needs to occur.**
- **Can be used as a list, stack or queue**

Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection c)	It is used to construct a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator.

Methods of LinkedList

Method	Description
boolean add(E e)	It is used to append the specified element to the end of a list.
void add(int index, E element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

boolean addAll(int index, Collection c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void addFirst(E e) void offerFirst( )	It is used to insert the given element at the beginning of a list.
void addLast(E e) void offerLast( )	It is used to append the given element to the end of a list.
void clear( )	It is used to remove all the elements from a list.
boolean contains(Object o)	It is used to return true if a list contains a specified element.
E element( )	It is used to retrieve the first element of a list.
E get(int index)	It is used to return the element at the specified position in a list.
E getFirst( ) peekFirst( )	It is used to return the first element in a list.
E getLast( ) peekLast( )	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.
ListIterator<E> listIterator(int index)	It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list.
E pop( )	It pops an element from the stack represented by a list.

void push(E e)	It pushes an element onto the stack represented by a list.
E remove( )	It is used to retrieve and removes the first element of a list.
E remove(int index)	It is used to remove the element at the specified position in a list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element in a list.
E removeFirst( ) pollFirst( )	It removes and returns the first element from a list.
Boolean removeFirstOccurrence(Object o)	It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail).
E removeLast( ) pollLast( )	It removes and returns the last element from a list.
boolean removeLastOccurrence(Object o)	It removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.
Object[] toArray( )	It is used to return an array containing all the elements in a list in proper sequence (from first to the last element).
int size( )	It is used to return the number of elements in a list.

```
//Program to Demonstrate LinkedList
import java.util.*;
public class LinkedList1
{
public static void main(String args[])
{
LinkedList<String> ls = new
LinkedList<String>();
ls.add("B");
ls.add("C");
ls.add("D");
ls.add("E");
System.out.println("The List is " +ls);
ls.add(2,"F"); //add at index 2
System.out.println("The List after add is "
+ls);
ls.set(2,"R");
System.out.println("The List after set is "
+ls);
LinkedList<String> ls1 = new
LinkedList<String>();
System.out.println("List after adding
other list is " +ls);
LinkedList<String> ls2 = new
LinkedList<String>();
ls2.add("X");
ls2.add("Y");
ls.addAll(4,ls2);
System.out.println("List after adding
other list at given index is " +ls);
ls.add("D");
ls.add("R");
ls.addFirst("A");
ls.addLast("Z");
```

```
System.out.println("The first
element of the list is "
+ls.element());
System.out.println("The first element of
the list is " +ls.getFirst());
System.out.println("The element at 6
place is "+ls.get(5));
System.out.println("The Last element of
list is " +ls.getLast());
System.out.println("The index of F
element is "+ls.indexOf("F"));
ls.push("M");
System.out.println("The List after push is
" +ls);
System.out.println("The Popped Element
is " +ls.pop());
ls.remove();
ls.remove(3);
ls.remove("P");
ls.removeFirstOccurrence("D");
ls.removeLastOccurrence("R");
System.out.println("The size of the list
finally : " +ls.size());
Iterator i = ls.iterator();
System.out.println("The Linked List
elements are:");
while (i.hasNext())
{
System.out.println(i.next());
}
/* Spliterator<String> list =
ls.spliterator();
System.out.println("list of letters using
spliterator:");
// forEachRemaining method of Spliterator
list.forEachRemaining((n) ->
System.out.println(n)); */
ls.clear();
System.out.println("The list after clear
"+ls);
```

3. HashSet class

HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

- **Stores the elements by using a mechanism called hashing.**
- **Contains unique elements only.**
- **Allows null value.**
- **Non synchronized.**
- **HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.**
- **H**

Constructors of Java HashSet class

Constructor	Description
HashSet( )	It is used to construct a default HashSet.
HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.

Methods of Java HashSet class

Modifier & Type	Method	Description
boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
void	clear( )	It is used to remove all of the elements from the set.
boolean	contains(Object o)	It is used to return true if this set contains the specified element.
boolean	isEmpty()	It is used to return true if this set contains no elements.

Iterator<E>	iterator()	It is used to return an iterator over the elements in this set.
boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.
int	size()	It is used to return the number of elements in the set.
Spliterator<E>	spliterator()	It is used to create a late-binding and fail-fast Spliterator over the elements in the set.
public boolean	addAll(Collection c)	It is used to insert the specified collection elements in the invoking collection.
public boolean	removeAll(Collection c)	It is used to delete all the elements of the specified collection from the invoking collection.
public void	clear()	It removes the total number of elements from the collection.
public boolean	containsAll(Collection c)	It is used to search the specified collection in the collection.

```
/* Use HashSet methods to perform operations on collection of data */
import java.util.*;
class HashDemo
{
public static void main(String args[])
{
//Creating HashSet
HashSet<String> set=new HashSet<String>();
//Adding Elements to HashSet - ignores duplicates
set.add("hyderabad");
set.add("hyderabad");
set.add("bangalore");
set.add("chennai");
set.add("kolkata");
set.add("kolkata");
set.add("pune");
System.out.println("The number of cities is " + set.size());
if(set.contains("chennai"))
System.out.println("Tamil Language");
System.out.println(i.next());
}
//Removing specific element from HashSet
set.remove("hyderabad");
//Displaying set
System.out.println("\n The list after remove ->" +set);
//adding another set
HashSet<String> set1=new HashSet<String>();
set1.add("Panjab");
set1.add("Delhi");
set.addAll(set1);
System.out.println("\n Updated after adding List is -> "+set);
//removing new set from list
set.removeAll(set1);
System.out.println("\n Updated List after removing list is -> "+set);
//Removing all the elements available in the set
set.clear();
```

4. TreeSet class

TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in sorted ascending order.

- **Contains unique elements only like HashSet.**
- **Access and retrieval times are quiet fast.**
- **Doesn't allow null element.**
- **Non synchronized.**
- **Maintains ascending order.**

Constructors of Java TreeSet class

Constructor	Description
TreeSet()	It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set.
TreeSet(Collection c)	It is used to build a new tree set that contains the elements of the collection c.

Methods of Java TreeSet class

Method	Description
boolean add(E e)	It is used to add the specified element to this set if it is not already present.
boolean addAll(Collection c)	It is used to add all of the elements in the specified collection to this set.
Iterator descendingIterator()	It is used iterate the elements in descending order.
E floor(E e)	It returns the equal or closest least element of the specified element from the set, or null there is no such element.
SortedSet headSet(E toElement)	It returns the group of elements that are less than the specified element.
E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no

Iterator iterator()	It is used to iterate the elements in ascending order.
E lower(E e)	It returns the closest least element of the specified element from the set, or null there is no such element.
E pollFirst()	It is used to retrieve and remove the lowest(first) element.
E pollLast()	It is used to retrieve and remove the highest(last) element.
SortedSet subSet(E fromElement, E toElement))	It returns a set of elements that lie between the given range which includes fromElement and excludes toElement.
SortedSet tailSet(E fromElement)	It returns a set of elements that are greater than or equal to the specified element.
boolean contains(Object o)	It returns true if this set contains the specified element.
boolean isEmpty()	It returns true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void clear()	It is used to remove all of the elements from this set.
E first()	It returns the first (lowest) element currently in this sorted set.
E last()	It returns the last (highest) element currently in this sorted set.
int size()	It returns the number of elements in this set.

```
//Program to Demonstrate TreeSet
import java.util.*;
public class TreeDemo
{ public static void main(String args[])
{ TreeSet<String> ts = new
TreeSet<String>();
ts.add("M"); ts.add("G");
ts.add("C"); ts.add("Z");
ts.add("W") ; ts.add("I");
System.out.println("The tree set is " +ts);
TreeSet<String> ts1 = new
TreeSet<String>();
ts1.add("A"); ts1.add("K");
ts.addAll(ts1);
System.out.println("The adding
collection is " +ts);
Iterator i = ts.iterator();
System.out.println("The Tree set elements
are:");
while (i.hasNext())
{
System.out.println(i.next());
} Iterator p = ts.descendingIterator();
while (p.hasNext())
{
System.out.println(p.next());
} System.out.println("The item lower than C
is " +ts.lower("C"));
System.out.println("The item higer than W is "
+ts.higher("W"));
System.out.println("The sub set is " +
ts.subSet("G" ,"M")); //G inclusive and M
exclusive
System.out.println("The list from I is "
+ts.tailSet("I"));
System.out.println("The list up to I is "
+ts.headSet("I"));
System.out.println("The first element of
list is " +ts.first());
System.out.println("The last element of list is
" +ts.first());
System.out.println("The first one removed "
+ts.pollFirst());
```

5. Date

The java.util.Date class represents date and time in java. It provides constructors and methods to deal with date and time in java.

java.util.Date Constructors

Constructor	Description
Date()	Creates a date object representing current date and time.
Date(long milliseconds)	Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.

java.util.Date Methods

Method	Description
boolean after(Date date)	tests if current date is after the given date.
boolean before(Date date)	tests if current date is before the given date.
int compareTo(Date date)	compares current date with given date.
boolean equals(Date date)	compares current date with given date for equality.
long getTime()	returns the time represented by this date object.
int hashCode()	returns the hash code value for this date object.
void setTime(long time)	changes the current date and time to given time.
String toString()	converts this date into Instant object.

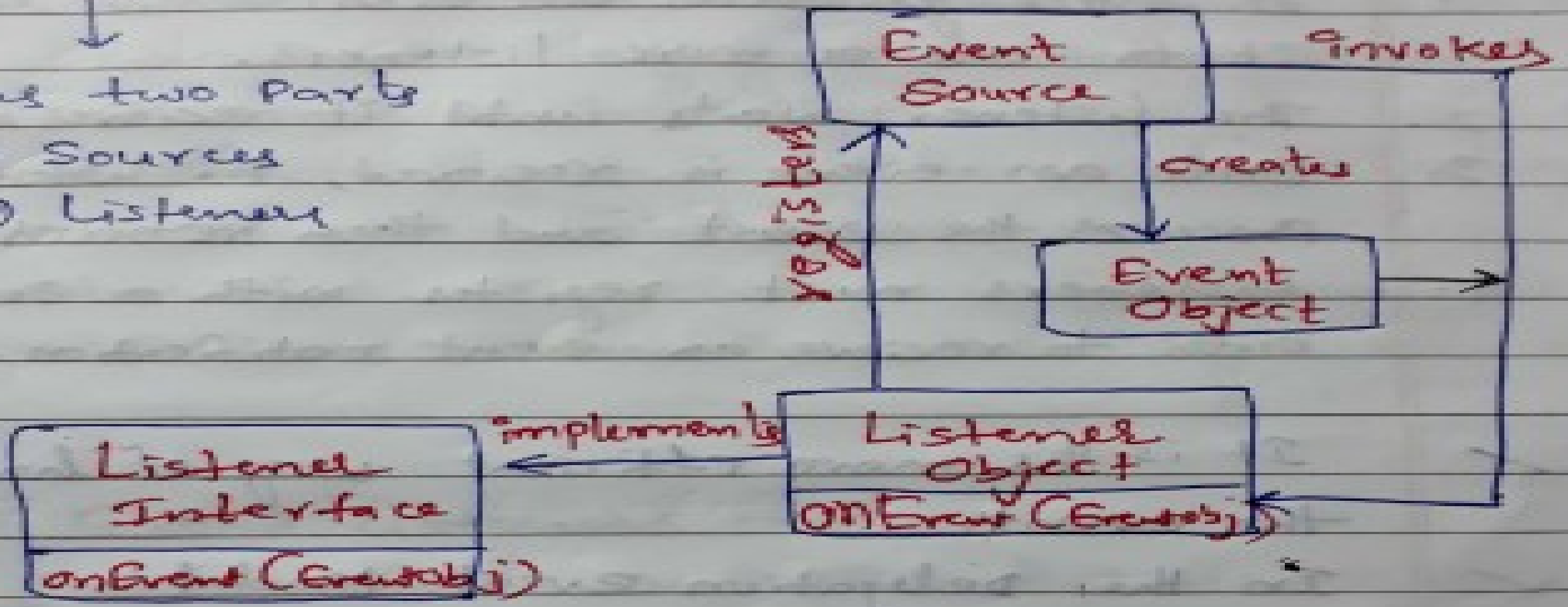
```
//Example Date class Methods
import java.util.Date;
public class DateDemo
{
public static void main(String args[])
{
// Instantiate a Date object
Date d1= new Date();
System.out.println("The current Date and
Time is " +d1.toString());
long msec = System.currentTimeMillis();
Date d2 = new Date(msec);
System.out.println("Using second
Constructor " + d2);
if(d1.equals(d2))
System.out.println("The Two Date objects
are equal");
else
System.out.println("The Two Date objects
are not equal");
System.out.println("Get the time of
Date d1 object " +d1.getTime());
System.out.println("Get the time of Date
d2 object " +d2.getTime());
int x = d2.compareTo(d1);
Date d3 = new Date();
d3.setTime(1256558992);
System.out.println("The Date object
d3 Date and Time is " +
d3.toString());
System.out.println("The dates
comparision is " +x);
if(d2.after(d3))
System.out.println("date3 is old");
else
System.out.println("date2 is old");
```

1. Delegation Event Model.

It has two parts

① Sources

② Listeners



— Java uses "delegation event model" to handle the events.

— Delegation Event Model defines standard and consistent mechanisms to generate and process events.

The Process of Delegation Event model is

1. A Source generates an event and sends it to one or more listeners.
2. The listener waits until it recieves an event.
3. Once an event is recieved, the listener Processes the event and then returns.
4. Listeners must register with a source in order to recieve an event notification.

- It is the more efficient way of handling the events.
- In the Delegation Event Model

1. Events — An event is an object that describes the state change in a source. Event occurs when we interact with GUI

Ex: clicking the mouse, key Press, button click, select an item in the list etc.

2. Event Sources —

- A source is an object that generates an event
- Each source may generate more than one type of Event.
- A source registers listener for receiving event for notification.

register ← public void addTypeListener(TypeListener el)

unregister ← public void removeTypeListener(TypeListener el)

3. Event Listeners —

- A listener is an object that is notified when an event occurs.
- A listener has to do
  1. Registered with one or more sources to recieve notifications about specific type of events
  2. Must implement methods to recieve and process the notifications.

→ A listener is an interface.

- All are defined in java.awt.event

2. Sources of Events

- Button — Generates action events when the button is pressed.
- Checkbox — Generates item events when the checkbox is selected or unselected.
- Choice — Generates item events when the item is changed
- List — Generates action events when the item is double clicked
  Generate item event when an item is selected or deselected
- Text Components — Generates text events when the (Textfield, user enters a character
  TextArea)
- Menus Item — Generates action event when a menus item is selected.
- Window — Generates window events when window is activated, closed, deactivated, opened or quit.

Page No.

- Scroll bar — Generate adjustment events when the scrollbar is manipulated.
- Applet — key & mouse events [Anything derived from Component class].

3. Event Classes

The package "java.awt.event" defines many types of events

ActionEvent — Generated when a button is pressed, a list item is double clicked, menu item is selected

AdjustmentEvent — Generated when a scrollbar is manipulated

ComponentEvent — Generated when a Component is hidden (or) moved (or) resized (or) becomes visible

ContainerEvent — Generated when a Component is added to (or) removed from a container.

FocusEvent — Generated when a Component gains or loses keyboard focus

ItemEvent — Generated when a checkbox (or) list item is clicked, also occurs when a choice selection is made etc.

KeyEvent – Generated when input is recieved from the keyboard.

MouseEvent – Generated when the mouse is dragged, moved, clicked, pressed or released, also generated when the mouse enters, exits a component

MouseWheelEvent – Generated when the mouse wheel is moved

TextEvent – Generated when the value of a text area or text field is changed.

WindowEvent – Generated when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

4. Event Listener

Date ___/___/___

Saathi

Interfaces

- Listeners are Interfaces
- Listeners are created by Implementing one or more interfaces defined by the java.awt.event package.
- When an event occurs, the event source invokes appropriate method defined by the listener and provides an event object as its argument.
- The various Listeners & their specific methods are

1. ActionListener — Recieve Action events.
   - void actionPerformed(ActionEvent ae)
2. AdjustmentListener — Recieve adjustment events
   - void adjustmentValueChanged(AdjustmentEvent ae)
3. ComponentListener — Recognize when a Component is hidden, moved, resized (or) shown.
   - void componentResized(ComponentEvent e)
   - void componentMoved(ComponentEvent e)
   - void componentShown(ComponentEvent e)
   - void componentHidden(ComponentEvent e)

4. ContainerListener — Recognize when a component is added or removed from the container.

void componentAdded (ContainerEvent e)
Void componentRemoved(ContainerEvent e)

5. FocusListener — Recognize when a component gains or loses a keyboard focus.

void focusGained (FocusEvent fe)
void focusLost (FocusEvent fe)

6. ItemListener — Recognize when the state of an item is changed.

void itemStateChanged (ItemEvent ie)

7. KeyListener — Regonize when a key pressed, released or typed.

void keyPressed (KeyEvent ke)
void keyReleased (KeyEvent ke)
void keyTyped (KeyEvent ke)

8. MouseListener — Recognises when the mouse (mouse) is clicked, enters a component, exits a component, is pressed, is released.

```
void mouseClicked (MouseEvent me)
void mouseEntered (MouseEvent me)
void mouseExited (MouseEvent me)
void mousePressed (MouseEvent me)
void mouseReleased (MouseEvent me)
```

9. MouseMotionListener — Recognizes when the mouse is dragged or moved.

```
void mouseDragged (MouseEvent me)
void mouseMoved (MouseEvent me)
```

10. MouseWheelListener — Recognise when the mouse wheel is moved.

```
void mouseWheelMoved (~~MouseEvent~~ MouseWheelEvent me)
```

11. TextListener — Recognizes when a text value changes in textfied (or) text area.

void textChanged (TextEvent te)

12. WindowFocusListener — Recognizes when a window gains or loses input focus.

void windowGainedFocus (WindowEvent we)
void windowLostFocus (WindowEvent we)

13. WindowListener — Recognizes when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

void windowActivated (WindowEvent we)
void windowClosed (WindowEvent we)
void windowClosing (WindowEvent we)
void windowDeactivated (WindowEvent we)
void windowDeiconified (WindowEvent we)
void windowIconified (WindowEvent we)
void windowOpened (WindowEvent we)

```
//Handling Mouse Events
import java.awt.*;
import java.awt.event.*;
class MouseFrame extends Frame
implements MouseListener,
MouseMotionListener
{
String msg="";
int X = 0, Y = 0;
MouseFrame()
{
addMouseListener(this);
addMouseMotionListener(this);
setSize(500,400);
//setLayout(null);
setVisible(true);
/* addWindowListener(new
WindowAdapter()
{
public void windowClosing(WindowEvent
e)
public void mouseClicked(MouseEvent
me)
{
X = 100;
Y = 50;
msg = "Mouse clicked.";
repaint();
}
public void mouseEntered(MouseEvent
me)
{
X = 100;
Y = 50;
msg = "Mouse entered.";
```

```
public void mousePressed(MouseEvent
me)
{
X = me.getX();//returns X co-ordinate
value
Y = me.getY();
msg = "Down";
repaint();
}
public void mouseReleased(MouseEvent
me)
{
X = me.getX();
Y = me.getY();
msg = "Up";
repaint();
}
public void mouseMoved(MouseEvent
me)
{
X = me.getX();
```

```
public void mouseDragged(MouseEvent
me)
{
X = me.getX();
Y = me.getY();
msg = "*****";
repaint();
}
public void paint(Graphics g)
{
g.drawString(msg,X,Y);
}
}
public class MouseDemo
```

```
//Handling Mouse Events
import java.awt.*;
import java.awt.event.*;
class KeyDemo extends Frame
implements KeyListener
{
String msg = "";
String msg1= "";
int X = 140, Y = 180; // output
coordinates
KeyDemo(String name)
{
super(name);
setForeground(Color.red);
addKeyListener(this);
/* addWindowListener(new
WindowAdapter()
{
public void windowClosing(WindowEvent
e)
{
public void keyPressed(KeyEvent ke)
{
msg1= "Key Down";
int key = ke.getKeyCode();
switch(key)
{
case KeyEvent.VK_F1 : msg += "<F1>";
break;
case KeyEvent.VK_F2 : msg += "<F2>";
break;
```

```
public void keyReleased(KeyEvent ke)
{
msg1="Key released";
repaint();
}
public void keyTyped(KeyEvent ke)
{
msg += ke.getKeyChar(); //gets the char
stroked
repaint(); }

public void paint(Graphics g)
{
Color c1 = new Color(123,50,89); //0 TO
255 rED,GREEN, BLUE
g.setColor(c1);
g.drawString(msg, X, Y);
g.drawString(msg1,100,200);
}
}

public class KeyDemo1
{
public static void main(String args[])
{
KeyDemo f = new KeyDemo("Key
Events");
f.setSize(300,400);
f.setVisible(true);
}
}
```

Adapter Classes

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

- **The adapter classes are found in java.awt.event and javax.swing.event packages.**
- **The A[illegible]s are given b[illegible]**

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

/Program to handle Window Event using Adapter Classes

```
import java.awt.*;
import java.awt.event.*;
public class AdapterDemo1 //extends
WindowAdapter
{
Frame f;
AdapterDemo1()
{
f=new Frame("Window Adapter");
f.addWindowListener();
public void windowClosing(WindowEvent
e)
{
f.dispose();
}
/* f.addWindowListener(new
WindowAdapter()
{
public void windowClosing(WindowEvent
e)
{
f.dispose();
}
}); */

f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}

public static void main(String[] args)
{
new AdapterDemo1();
}
```

```
//Program to handle Key Event using Adapter Classes
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterDemo extends KeyAdapter
{
Label l;
TextArea area;
Frame f;
KeyAdapterDemo()
{
f=new Frame("Key Adapter");
l=new Label();
l.setBounds(20,50,200,20);
area=new TextArea();
area.setBounds(20,80,300, 300);
area.addKeyListener(this);
f.add(l);f.add(area);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public void keyReleased(KeyEvent e)
{
String text=area.getText();
String words[]=text.split(" ");
l.setText("Words: "+words.length +"
Characters:"+text.length());
}
public static void main(String[] args)
{
new KeyAdapterDemo();
}
}
```

//Program to handle Mouse Event using Adapter Classes

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterDemo extends
MouseAdapter
{
Frame f;
MouseAdapterDemo()
{
f=new Frame("Mouse Adapter");
f.addMouseListener(this);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public void mouseClicked(MouseEvent e)
{
Graphics g=f.getGraphics();
g.setColor(Color.BLUE);
g.fillRect(e.getX(),e.getY(),30,30);
}

public static void main(String[] args)
{
new MouseAdapterDemo();
}
}
```